НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних»

Виконав:
студент IV курсу, групи КП-51
Андрієнко Федір Олегович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Заболотня Т.М. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н.,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____Дичка І.А.
«___»_____ 2018 р.

ЗАВДАННЯ

на дипломний проект студенту

Андрієнку Федору Олеговичу

1. Тема проекту «Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних», керівник проекту Заболотня Тетяна Миколаївна, доцент, к.т.н., затверджені наказом по університету від «22» травня 2019 р. №1331-С.

2. Термін подання студентом проекту «18» червня 2019 р.

3. Вихідні дані для дипломного проектування: див. Технічне завдання.

4. Перелік задач, які потрібно вирішити:
   - проаналізувати отримане структуроване питання;
   - розглянути існуючі рішення для семантичного пошуку;
   - розробити структуру пошуку в неоднорідному корпусі даних;
   - реалізувати алгоритм семантичного пошуку та обробки тексту природної мови;
   - виконати тестування програмної системи.

5. Перелік обов'язкового ілюстративного матеріалу:
   - структурна схема програмної системи (креслення);
   - структурна схема семантичного пошуку (креслення);
   - алгоритм семантичного пошуку в корпусі неоднорідних даних (плакат);
   - алгоритм обробки природної мови (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент, к.т.н.		

7. Дата видачі завдання «31» жовтня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Вивчення літератури за тематикою проекту	14.11.2018	
2.	Розробка та узгодження технічного завдання	28.11.2018	
3.	Підготовка матеріалів першого розділу дипломного проекту	15.12.2018	
4.	Розроблення семантичного пошуку	16.01.2019	
5.	Підготовка матеріалів другого розділу дипломного проекту	01.03.2019	
6.	Розроблення алгоритму семантичного пошуку в корпусі неоднорідних даних	22.03.2019	
7.	Програмна реалізація та тестування програмної системи	12.04.2019	
8.	Підготовка матеріалів третього розділу дипломного проекту	25.04.2019	
9.	Підготовка матеріалів четвертого розділу дипломного проекту	02.05.2019	
10.	Оформлення документації дипломного проекту	25.05.2019	

Студент _____ Андрієнко Ф.О.

Керівник проекту _____ Заболотня Т.М.

АНОТАЦІЯ

Даний дипломний проект присвячено створенню програмного модулю для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних, з використанням засобів обробки природної мови.

Тема роботи обумовлена необхідністю зниження часу, який витрачається при пошуку інформації в різних пошукових системах та автоматизації цього процесу.

Розроблена програмна система являє собою сервер для виконання семантичного пошуку в корпусі неоднорідних даних та обробку вже знайдених фрагментів тексту природної мови. Природна мова застосовується після того, як були знайдені інформаційні фрагменти тексту, щоб покращити відповіді на структуровані питання та зменшити фрагменти тексту, зробивши їх більш релевантними та зручними для читання кінцевому користувачу.

Доступ до функціональності системи забезпечується за допомогою іншого модуля, який безпосередньо являється інтерфейсом для користувача та аналізом запиту цього користувача, за допомогою обробки природної мови. Інші способи доступу до функціоналу системи, окрім як через описаний модуль вище, відсутні, однак наявна архітектура дає можливість додавати інші модулі для взаємодії з модулем семантичного пошуку.

У даному дипломному проекті розроблено: архітектуру системи, алгоритм семантичного пошуку на структуровані питання в неоднорідному корпусі даних та обробку тексту за допомогою природної мови, знайденого семантичним пошуком.

ABSTRACT

This diploma project is devoted to the creation of a software module for the semantic search for answers to structured questions in the body of heterogeneous data, using natural language processing tools.

The theme of the work is due to the need to reduce the time spent in the search for information in various search engines and automate this process.

The developed software system is a server for performing semantic search in the body of heterogeneous data and processing of already found fragments of natural language text. The natural language is used after the text pieces have been found to improve the responses to structured questions and to reduce the fragments of the text by making them more relevant and user-friendly to read to the end user.

Access to the functionality of the system is provided by another module, which is directly a user interface and analyzes the request of this user, using natural language processing. There are no other ways to access the functional system, except through the described module above, but the existing architecture enables the addition of other modules to interact with the semantic search module in the body of heterogeneous data.

This graduation project has developed: the architecture of the system, the algorithm of semantic search on structured issues in a non-uniform data enclosure and the processing of text using the natural language found by semantic search.

ДП.045440-01-90 Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних. Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Програмний модуль для	5	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Технічне завдання		
ДП.045440-03-81	Програмний модуль для	46	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Пояснювальна записка		
ДП.045440-04-51	Програмний модуль для	4	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Програма та методика		
	Тестування		
ДП.045440-05-34	Програмний модуль для	3	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Програмний модуль для	1	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Структурна схема		
	програмної системи.		
	UML діаграма		
	компонентів.		
ДП.045440-07-99	Програмний модуль для	1	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Структура бази даних.		
	ER діаграма		
ДП.045440-08-98	Програмний модуль для	1	
	семантичного пошуку		
	відповідей на структуровані		
	питання у корпусі		
	неоднорідних даних.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНИЙ МОДУЛЬ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ВІДПОВІДЕЙ НА СТРУКТУРОВАНІ ПИТАННЯ У КОРПУСІ НЕОДНОРІДНИХ ДАНИХ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Ф.О. Андрієнко

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для допомоги студентам у знаходженні/пошуку найбільш релевантної та корисної інформації у межах певної предметної дисципліни (за приклад буде взята дисципліна "Основи програмування") на просторах глобальної мережі Інтернет та текстових онлайн документів (наприклад, створених викладачем лекцій).

Також, в призначення розробки програмного модуля входить наповнення бази даних відповідей, які були найбільш ефективними та корисними для користувача (студента).

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмний модуль повинен забезпечувати такі основні функції:

1. Аналіз запиту:
   - Запити: структуровані;
   - Локалізація: англійська мова;

- Структура запиту: текстова;
- Наявність контексту запиту.

2. Задача пошуку:
   - Розмір корпусу даних: буде складати приблизно тисячі спеціалізованих веб-документів;
   - Носій інформації: пошук по тексту;
   - Контроль і якість корпусу даних: неоднорідний корпус даних, містить як і готові до індексації документи (веб-сторінки та онлайн-словники) так і сирі документи (“Google Документи”);
   - Швидкість індексації: у реальному часі;
   - Затримка: 10 секунд.

Розробку виконати на мові програмування *Python.*

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структурна схема програмної системи. UML діаграма компонентів»;
   - «Структура бази даних питань та відповідей. ER діаграма».

6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2018

Розроблення та узгодження технічного завдання……....28.11.2018

Підготовка матеріалів першого розділу дипломного проекту……..……..……15.12.2018

Розроблення семантичного пошуку…….....16.01.2019

Підготовка матеріалів другого розділу дипломного проекту...…….....……....01.03.2019

Розроблення алгоритму семантичного пошуку в корпусі неоднорідних даних…….....22.03.2019

Програмна реалізація та тестування програмної системи……..……..….12.04.2019

Підготовка матеріалів третього розділу дипломного проекту …….…….....25.04.2019

Підготовка матеріалів четвертого розділу дипломного проекту …….....02.05.2019

Оформлення документації дипломного проекту…….25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ПРОГРАМНИЙ МОДУЛЬ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ВІДПОВІДЕЙ НА СТРУКТУРОВАНІ ПИТАННЯ У КОРПУСІ НЕОДНОРІДНИХ ДАНИХ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____Ф.О. Андрієнко

2019

ЗМІСТ

ВСТУП

Кількість інформації, яку створює світова спільнота, зростає з кожним роком. Відкритість інформаційного поля теоретично забезпечує вільний і швидкий доступ до даних. Однак у такої загальної доступності є і зворотня сторона – щоб отримати інформацію, її потрібно спочатку знайти.

Отримання інформації, яка цікавить користувача, в переважній більшості випадків зводиться до використання інтернет ресурсів пошукових систем (ПС).

Пошук інформації пошуковим роботом являє собою процес виявлення в індексованих документах релевантних фрагментів тексту, тобто таких, які задовольняють заздалегідь сформований запит.

Основне завдання полягає в тому, щоб на конкретний структурований запит користувача ПС провела обробку інформації з подальшим ранжуванням знайдених даних по їх релевантності.

Користувач не може описати системі ознаки шуканого об'єкта, оскільки принцип пошуку ПС базується на тексті і ключових словах. Фактично користувачеві складно знайти дані, про які він ще не знає.

Таким чином, основною проблемою знаходження смислової відповідності документа запиту користувача є розробка і реалізація підходів, заснованих на семантичному пошуку.

Даний модуль призначений для допомоги студентам, які ще невпевнено володіють пошуковими системами.

Головною проблемою, з якою стикаються користувачі – це знаходження найбільш релевантної та корисної інформації у межах певної предметної дисципліни (за приклад буде взята дисципліна “Основи програмування”) на просторах глобальної мережі Інтернет.

Найчастіше студенти звертаються до викладача через різні види месенджерів, щоб дізнатися відповідь на потрібні їм питання. Але буває так, що викладач не завжди встигає оперативно відповісти на поставлене запитання, тому студент змушений шукати потрібну йому інформацію на просторах Інтернету самостійно, і для пришвидшення рішення проблеми студент може легко використати даний модуль.

Проблема в тому, що список літератури та посилання на ресурси, які дають студентам викладачі дуже великий, тому даний модуль пришвидшує пошук у цьому корпусі неоднорідних даних.

Даний модуль призначений на знаходження найбільш релевантної інформації в певній предметній області. Тому побудова такого модуля, який буде автоматично надавати відповіді студентам на задані ними питання з певної дисципліни – це актуальна задача.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

БД *(DB, Database)* – база даних;

ОС – операційна система;

КПІ ім. Ігоря Сікорського – Київський Політехнічний Інститут імені Ігоря Сікорського;

ПЗ – програмне забезпечення, програмний застосунок;

ПС – пошукова система;

SW (Semantic Web) – розширення Всесвітньої павутини через стандарти Всесвітнього консорціуму (W3C). Стандарти пропагують загальні формати даних і обмінні протоколи в Інтернеті, найбільш фундаментально RDF. Згідно з W3C, Semantic Web надає загальну структуру, яка дозволяє спільно використовувати дані в межах програми, підприємства і громади. Таким чином, семантичний веб розглядається як інтегратор у різних інформаційних додатках та системах;

W3C (World Wide Web Consortium) – є головною міжнародною організацією стандартів для World Wide Web (скорочено WWW або W3). Заснований і в даний час очолює Тім Бернерс-Лі, консорціум складається з організацій-членів, які працюють на постійній основі з метою спільної роботи над розробкою стандартів Всесвітньої мережі. Станом на 29 травня 2019 р. Консорціум Всесвітньої павутини (W3C) має 444 члени. W3C також займається освітою та інформаційною підтримкою, розробляє програмне забезпечення та служить відкритим форумом для обговорення Інтернету;

RDF (Resource Description Framework) – є сімейством специфікацій World Wide Web Consortium (W3C), спочатку розроблених як модель даних метаданих. Він став використовуватися в якості загального методу для концептуального опису або моделювання інформації, яка реалізована в веб-

ресурсах, використовуючи різні синтаксичні позначення і формати серіалізації даних. Він також використовується в програмах управління знаннями;

OWL (Ontology Web Language) – є сімейством мов для авторських онтологій. Онтології нагадують ієрархії класів в об'єктно-орієнтованому програмуванні, але є кілька критичних відмінностей. Ієрархії класів призначені для представлення структур, що використовуються у вихідному коді, які розвиваються досить повільно (зазвичай щомісячні ревізії), тоді як онтології повинні представляти інформацію в Інтернеті і, як очікується, розвиватимуться майже постійно. Аналогічно, онтології, як правило, набагато більш гнучкі, оскільки вони призначені для представлення інформації в Інтернеті, яка походить від різноманітних гетерогенних джерел даних. Ієрархії класів, з іншого боку, повинні бути досить статичними і спиратися на набагато менш різноманітні та структуровані джерела даних, такі як корпоративні бази даних;

SWD (Semantic Web Document) – передбачає публікацію мов, спеціально розроблених для даних: Resource Description Framework (RDF), Web Ontology Language (OWL) і Extensible Markup Language (XML);

Q&A System (Questions and Answers System) – інформаційна система, яка виконує обробку питань природною мовою. Аналізуючи запити, система надає на них пряму відповідь, знайдену у певному сховищі даних, або генерує її, використовуючи інформацію з декількох джерелах інформації;

API (*Application Programming Interface*, прикладний програмний інтерфейс) – набір функцій, методів, протоколів взаємодії з певним програмним застосунком;

AI (Artificial Intelligence) – штучний інтелект;

DL (Deep Learning) – глибоке навчання;

NLP (Natural Language Processing) – обробка природної мови (тексту);

Back End – це серверна частина, яка виконує бізнес логіку та є пластом доступу до даних;

Front End – це клієнтська частина, абстракція над пластом *Back End*, яка надає доступ до розташованого за ним функціоналу через інтерфейс, зручний для користувача;

POS tagging (Part-of-speech tagging) – це розмітка текстових даних; пов’язування слів з текстового набору з відповідними ним частинами мови.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз особливостей корпусу неоднорідних даних

Якщо відкрити будь-яку веб-сторінку у браузері, то вона буде складатися з великого набору тегів та атрибутів, які міститимуть у собі текст.

Цей текст вже безпосередньо відображається у браузері за допомогою тегів. Але щоб текст коректно відображався на веб-сторінці потрібно правильно оперувати HTML-розміткою.

Кожен HTML тег відповідає за певне відображення та розташування на веб-сторінці.

Рис. 1. Приклад корпусу неоднорідних даних (як це бачить користувач веб-ресурсів)

Семантичний пошук є одним з методів інформаційного пошуку і являє собою процес пошуку документів по їх смисловому змісту. Основою семантичного пошуку служать заздалегідь встановлені відносини між символами і об'єктами, які вони позначають.

Можна виділити два основних види семантичного пошуку. Повнотекстовий пошук – пошук по всьому вмісту документа з використанням

попередньо побудованих індексів. Пошук по метаданих – це пошук за деякими атрибутами документа, які описують певні об'єкти підтримувані системою, наприклад, автор, адреса, назва організації і т. д. Саме використання метаданих сьогодні широко застосовується в інтернеті під назвою SW. Основний акцент концепції робиться на роботі з метаданими, які однозначно характеризують властивості і зміст веб-ресурсів, замість використовуваного в даний час текстового аналізу документів. Ця концепція була прийнята і просувається Консорціумом W3C. Для її впровадження передбачається створення мережі документів, що містять метадані про web-ресурс. Тоді як самі ресурси призначені для сприйняття людиною.

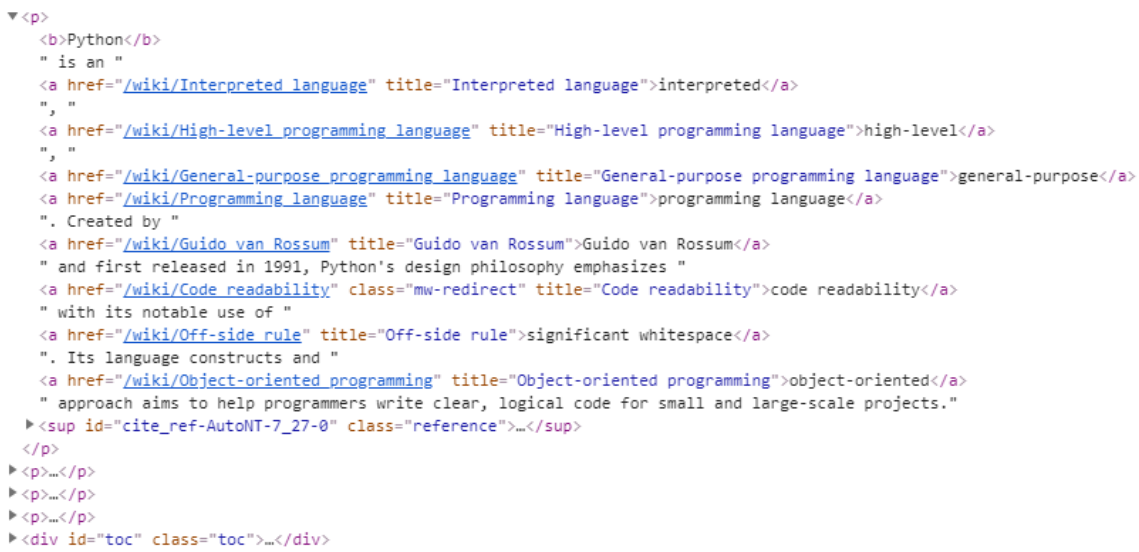


Рис. 2. Приклад корпусу неоднорідних даних (як це представлено на програмному рівні)

Пошукові системи, які здійснюють пошук за ключовими словами, забезпечують доступ до мільярдів індексованих Інтернет-сторінок для мільярдів користувачів. Такі явища, як полісемія (одне слово має кілька значень) і синонімія слів (кілька слів з одним значенням) збільшують число

нерелевантних результатів, що видаються пошуковою системою. У зв'язку з постійно зростаючою кількістю сайтів зростає потреба в ретельному аналізі контенту Інтернет-документів для того, щоб звести можливість отримання нерелевантних результатів до мінімуму. Технології семантичної павутини надають можливості для вирішення цієї проблеми.

Семантична павутина (Semantic Web) є розширенням традиційного Інтернету і націлена на спрощення пошуку і розподілу інформації. Дана технологія грунтується на елементах, побудованих з використанням стандартних мов онтологій, таких як OWL. Звичайні пошукові системи ґрунтуються на пошуку ключових термінів запиту в документі і не можуть використовувати його смислове значення для отримання результату, тому співтовариство дослідників семантичної павутини запропонувало використовувати семантичні пошукові технології, серед яких OntoSearch, Semantic Portals, Semantic Wikis, мультиагент P2P, семантичні системи маршрутів (запитів), питально-відповідні системи, що використовують онтології для зберігання баз знань [1].

Документ семантичної павутини SWD можна розглядати як набір даних, контентом якого є звичайний документ, розмічений певними тегами, взятими з предметної області. Такі Інтернет-документи можуть бути розподілені по безлічі різних категорій, що відносяться до типів, використовуваних для розмітки документа.

Розглянемо існуючі технології семантичної павутини в контексті наступних проблем: автоматичне створення формального запиту (онтології запиту) і отримання колекцій документів, структура яких невідома заздалегідь (розподілені і семантично неоднорідні дані).

Стандартний Інтернет-пошук за ключовими словами базується на пошукових технологіях, основою яких є виявлення строкової (лексичної) відповідності запитуваних термінів, які мають Інтернет-документи. Зазвичай

Інтернет-пошук по ключу застосовується для пошуку неструктурованих Інтернет-документів (текст без семантичної розмітки).

Найбільш популярним видом Інтернет-пошуку є логічний пошук, заснований на виявленні комбінацій ключових слів, розділених операторами AND, OR, NOT.

1.2. Аналіз існуючих систем пошуку відповідей на питання

В рамках виконання даного дипломного проекту також досліджено можливості існуючих програмних засобів, які можуть проводити пошук. В основному, такі системи призначені для надання відповідей на задані ним запити-питання. Метою проведеного огляду було дослідити існуючі програмні продукти, призначені для автоматичного надання відповідей, і з’ясувати, наскільки вони відповідають вказаним в задачі вимогам.

1.2.1. Siri

Siri є персональним помічником Apple для пристроїв iOS, macOS, tvOS і watchOS, які використовують функцію розпізнавання голосу та штучний інтелект (AI).

Siri реагує на запитання користувачів та надає відповідну інформацію на головному екрані з певних програм, таких як веб-пошук або календар. Служба також дозволяє користувачам диктувати електронні листи та текстові повідомлення, читає отримані повідомлення електронної пошти та повідомлення та виконує різні інші завдання.

Siri може виконувати різні завдання, такі як:

- навігація в напрямку;
- планування подій та нагадування;
- пошук в Інтернеті;
- інформування про ретрансляцію;

- зміна параметрів пристрою, який використовує можливості Siri;
- обробка параметрів природною мовою.

Функції Siri розміщені головним чином у хмарі, включаючи головне автоматичне розпізнавання мови та інтерпретацію природного мови. Проте невеликий розпізнавач мовлення постійно працює на локальному пристрої, слухаючи слова "Hey Siri". Коли розпізнавач мовлення виявляє фразу, він використовує глибоку нейронну мережу для перетворення акустичного малюнка голосу користувача в розподіл ймовірностей.

Потім, детектор "Hey Siri" об'єднує безперервний потік інформації з цієї моделі в окремі події для розрахунку оцінки впевненості, що ця фраза дійсно була "Hey Siri". Siri активується, якщо оцінка досить висока.

Коли користувач говорить команду або запит, Siri збирає звук, перетворює його у файл даних і надсилає його на сервери Apple. Пристрій повинен бути підключений до Інтернету для функціонування Siri. Як тільки голосові дані знаходяться на серверах, вони проходять через безліч блок-схем, утворених великою базою запитань і відповідей.

1.2.2. Bixby

Bixby – це система AI, яка спроектована, щоб полегшити взаємодію з пристроєм, зокрема, щоб уникнути складності пристроїв, які все частіше використовуються. Вона дебютувала на пристроях Samsung Galaxy S8 і Note 8, але розрахований на роботу в різних продуктах Samsung.

За своєю суттю, ви можете використовувати її для тексту, отримувати спеціальну інформацію (про погоду, нагадування про зустрічі, статті про новини тощо).

Bixby може вивчати індивідуальні голоси, так що вона буде персоналізувати відповіді в залежності від того, хто запитує.

Компанія Samsung заявила, що вона "навчається, розвивається і адаптується" до вас.

У 2017 році Samsung оголосила про версію другого покоління Bixby. За даними компанії, Bixby 2.0 є "фундаментальним стрибком вперед для цифрових помічників" і "сміливим переосмисленням платформи". У двох словах: Bixby 2.0 призначена для того, щоб зробити Bixby доступним на "будь-якому пристрої". Це також "відкрито", що дозволяє розробникам інтегрувати голосовий помічник у свої продукти та вибирати, як ви будете взаємодіяти зі своїми додатками.

Компанія Samsung розповіла про те, що існує три стовпи, які допомагають запустити Bixby:

- Bixby – це комплексне рішення: він призначений для виконання повного спектру взаємодій, замість запуску програми, наприклад, або виконання одного завдання. Samsung говорить, що Bixby зможе робити все, що можна робити з додатком, використовуючи дотик;
- Bixby – це один з словників AI, продемонстрований, наприклад, Google Assistant. Це означає, що Bixby може розпізнати стан, в якому знаходиться програма, і робити правильні дії на основі ваших запитів, а також дозволяє змішувати голос або дотик;
- Bixby розуміє природну мову: це означає, що вам не потрібно використовувати набір фраз, але ви можете дати неповну інформацію, і Bixby може інтерпретувати і вжити заходів. Природне розпізнавання мови було ключем до зростання Alexa, наприклад, і зараз є ключовим елементом сучасного AI.

Служба по суті працює так само, як і інші рішення AI, такі як Google Assistant або Amazon Alexa в тому, що вона слухає ваш голос, інтерпретує інформацію та повертає отриману дію.

1.2.3. Google Assistant

Google випереджає Alexa, Siri і Bixby своїм власним голосовим помічником: Google Assistant. Це зробило неймовірний прогрес з його запуску 2016 та є певно найбільш передовим та динамічний асистент.

Через три роки після запуску Google Assistant Google розповсюдив асистента не тільки на власному апаратному забезпеченні Google, але й через партнерство з іншими компаніями.

Спочатку Google Now уміло витягувала для вас релевантну інформацію: вона знала, де ви працювали, ваші зустрічі і плани подорожей, спортивні команди, які вам сподобалися, і те, що вас цікавило, щоб він міг представити вам особисту інформацію, яка мала значення.

Google Асистент підтримує як текстовий, так і голосовий запис "ОК Google" або "Hey, Google" охоплює голосові команди, голосовий пошук і керування пристроєм, що активується голосом, дозволяючи робити нагадування, надсилати повідомлення, перевіряти зустрічі і так далі, як на пристроях Android, так і на Apple.

Асистент Google:

- можливість керувати своїми пристроями та розумним будинком;
- має доступ до інформації з календарів та іншої особистої інформації;
- може знайти інформацію в Інтернеті, від бронювання ресторанів до напрямків польоту літаків;
- надає можливість керувати своєю музикою;
- може відтворювати вміст вашого Chromecast або інших сумісних пристроїв;
- може запускати таймери та нагадування;
- надає можливість перекладати в режимі реального часу.

Google також може розпізнавати голосові профілі для різних людей, тому він знає, хто з ним розмовляє і може відповідно адаптувати відповіді – те, що інші системи тільки починають реалізовувати.

Також є можливість запитати декілька речей одночасно. У лінгвістиці це називається координаційним скороченням. Оволодіння запитами, подібними до цього, є, ймовірно, тим, що допоможе Google Assistant випередити конкурентів.

Оскільки помічник Google знає вас і розуміє контекст, він може бути більш швидким у пошуку інформації, яка необхідна вам.

У майбутньому Google навіть каже, що помічник зможе дзвонити і бронювати зустрічі для вас. Він розроблений для того, щоб бути більш ніж просто реактивним.

Особливо подобається новий режим інтерпретатора, який почав розгортатися на початку 2019 року на пристроях Google Home і смарт-дисплеях. З його допомогою ви можете попросити Google Assistant допомогти вам у проведенні розмов на десятках мов.

Асистент Google у домашніх пристроях Google формує основу управління розумним будинком. Він сумісний з широким спектром пристроїв,

завдяки чому ви можете керувати підігрівом, підсвічуванням та багато іншого за допомогою голосу.

1.2.4. Google

Google Search, який також називається Google Web Search або просто Google, – це веб-пошукова система, розроблена компанією Google LLC. Це найпопулярніший пошуковий механізм у Всесвітній павутині на всіх платформах, з 92,74% ринкової частки станом на жовтень 2018 року, що обробляє більше 3,5 мільярдів пошуків щодня.

Пошук базується, зокрема, на системі рангу пріоритетів, яка називається "PageRank". Пошук Google також надає різні варіанти налаштування пошуку, використовуючи символи, які включають, виключають, визначають або вимагають певної поведінки пошуку, а також надають спеціалізований інтерактивний досвід, наприклад, статус рейсу та відстеження пакетів, прогноз погоди, конверсії валюти, одиниці та часу визначення та багато іншого.

Основною метою пошуку Google є пошук тексту в загальнодоступних документах, пропонованих веб-серверами, на відміну від інших даних, таких як зображення або дані, що містяться в базах даних.

Аналіз частоти пошукових термінів може вказувати на економічні, соціальні та медичні тенденції.

У США станом на липень 2018 року сайти Microsoft обробили 24,2% всіх пошукових запитів у Сполучених Штатах. Протягом того самого періоду часу Oath (раніше відома як Yahoo) мала частку пошукового ринку 11,5%.

Лідер ринку Google створив 63,2 відсотка всіх основних пошукових запитів у Сполучених Штатах.

Принципи роботи Google Пошуку:

- Сканування та індексування: Google використовує веб-сканери, щоб зчитувати інформацію веб-сторінок та іншого загальнодоступного контенту в пошуковому індексі.
- Щоб користувачі могли швидко знайти потрібні відомості, роботи збирають інформацію на сотнях мільярдів сторінок і впорядковують її в пошуковому індексі.
- При черговому скануванні поряд зі списком веб-адрес, отриманих під час попереднього сканування, використовуються файли Sitemap, які надаються власниками сайтів. У міру відвідування сайтів робот переходить за вказаними на них посиланнях на інші сторінки. Особливу увагу він приділяє новим і зміненим сайтам, а також непрацюючим посиланням. Він самостійно визначає, які сайти сканувати, як часто це потрібно робити і скільки сторінок слід вибрати на кожному з них. Власники сайтів можуть за допомогою інструментів для веб-майстрів вказувати, як саме слід сканувати їх ресурс, зокрема, надавати докладні інструкції по обробці сторінок, запитувати їх повторне сканування, а також забороняти сканування, використовуючи файл robots.txt. Google не збільшує частоту сканування окремих ресурсів за плату. Власникам всіх сайтів доступні однакові інструменти, що дозволяють забезпечити високу якість результатів пошуку по їх сторінках.
- Інтернет схожий на бібліотеку, яка містить мільярди видань і постійно поповнюється, але не має в своєму розпорядженні централізованою системою обліку. Щоб знаходити загальнодоступні сторінки, ми використовуємо спеціальне програмне забезпечення, зване пошуковими роботами.

- Роботи аналізують сторінки і переходять по посиланнях на них – як звичайні користувачі. Після цього вони надсилають відомості про ресурси на сервери Google.
- Під час сканування системи обробляють матеріали сторінок так само, як це роблять браузери, і реєструють дані за ключовими словами і новизні контенту, а потім створюють на їх основі пошуковий індекс. Пошукової індекс Google містить сотні мільярдів сторінок. Його обсяг – більше 100 млн ГБ. Він нагадує сторінку зі змістом книги, так як в ньому є окремий запис по кожному слову на всіх проіндексованих сторінках. Під час індексування дані по сторінці додаються в запису по всьому словами, які вона містить.
- Алгоритми Google Пошуку: щоб користувачі отримували актуальні і релевантні запитам результати, інструменти ранжирування Google впорядковують сотні мільярдів веб-сторінок в пошуковому індексі.
- Щоб користувачі за секунди отримували не нескінченні списки адрес, а актуальні і релевантні результати, системи ранжування Google впорядковують сотні мільярдів сторінок в пошуковому індексі. Ці системи ранжування складаються з наборів алгоритмів, які, завдяки постійній оптимізації Google Пошуку, все більш точно визначають, що цікавить користувачів і які результати слід показати.
- Аналіз слів і виразів: щоб підібрати сторінки, що містять релевантні відомості, перш за все необхідно проаналізувати значення слів в запиті. Google розробляє мовні моделі, що дозволяють визначати, які поєднання слів слід шукати в індексі. Для цього виконується ряд дій – від інтерпретації орфографічних помилок до визначення типу введеного запиту на основі результатів останніх досліджень в області розуміння природної мови.

- Підбір відповідних сторінок: йде підбір сторінок, що містять інформацію, яка відповідає запиту. Зазвичай, коли користувач вводить запит, алгоритми шукають в індексі відповідні сторінки, а також визначають, як часто ключові слова зустрічаються на сторінці і в яких її розділах (наприклад, в заголовку або основному тексті). Алгоритми не тільки зіставляють ключові слова, а й визначають, наскільки повна інформація міститься в результаті пошуку.
- Ранжування релевантних сторінок: у більшості випадків інформацію, яка відповідає запиту, містять тисячі або навіть мільйони сторінок. Алгоритми, що дозволяють оцінювати релевантність сторінок, щоб найбільш підходящі з них показувалися першими. Щоб надавати найбільш актуальну інформацію, ці алгоритми оцінюють сотні самих різних чинників – від новизни контенту і кількості повторів запиту до зручності перегляду сторінки.
- Показ найбільш потрібних результатів: перш ніж показувати результати пошуку, йде оцінка всієї знайденої інформацію в комплексі. Це дозволяє визначити, чи йде мова про одну тему або декількох і не відноситься більшість сторінок до надмірно вузької трактуванні пошукового запиту. Таким чином пошуковик прагне надавати користувачам різноманітну інформацію в найбільш зручній для них формі. У міру розвитку Інтернету йде оптимізація свої системи ранжування так, щоб результати по максимальному числу запитів були якомога більш релевантними.
- Облік відомостей про користувачів: щоб надавати користувачам найбільш підходящу і актуальну інформацію, пошуковик враховує відомості про їх місцезнаходження, попередніх запитах, налаштуваннях Google Пошуку і т.д. Пошуковик показує результати

пошуку, виходячи з відомостей про країну і місце розташування користувачів.

- Корисні відповіді: обсяги даних і різноманітність матеріалів в Інтернеті постійно зростають. Google представляє результати пошуку в різних форматах, щоб користувачі якомога швидше отримували потрібні відомості.

1.3. Аналіз вимог до функціональності програмного модуля

Пошукові системи, що використовують пошук по метаданих, працюють з об'єктами, а не з фрагментами тексту, а, отже, подібний підхід дозволяє здійснювати більш ефективний пошук. Однак слабкість такого підходу полягає в тому, що зараз практично вся інформація в інтернеті являє собою як раз гіпертекст з розміткою, а саме неоднорідні дані, і, щоб настільки ж ефективно вирішувати завдання глобального пошуку, потрібно навчитися з тексту виділяти об'єкти.

Використовуючи технології NLP можна досягти бажаного початкового результату.

Процес читання і розуміння англійського тексту сам по собі дуже складний. Крім того, люди часто не дотримуються логіки і послідовності розповіді. Реалізація будь-якого складного комплексного завдання в машинному навчанні зазвичай означає побудову конвеєра.

Сенс цього підходу в тому, щоб розбити проблему на дуже маленькі частини і вирішувати їх окремо. Поєднавши кілька таких моделей, що поставляють один одному дані, ми можете отримувати чудові результати.

Розбиття процесу мовного аналізу на етапи, у ході яких на виході буде отримано максимально чітке та зрозуміле речення, а не великий фрагмент тексту.

Програмний модуль повинен забезпечувати такі основні функції:

- Аналіз запиту:
    - запити: структуровані;
    - локалізація: англійська мова;
    - структура запиту: текстова;
    - наявність контексту запиту.
- Задача пошуку:
    - розмір корпусу даних: буде складати приблизно тисячі спеціалізованих веб-документів;
    - носій інформації: пошук по тексту;
    - контроль і якість корпусу даних: неоднорідний корпус даних, містить як і готові до індексації документи (веб-сторінки та онлайн-словники) так і сирі документи (“Google Документи”);
    - швидкість індексації: у реальному часі;
    - затримка: 10 секунд.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ ТА СПОСОБІВ ВИКОНАННЯ СЕМАНТИЧНОГО ПОШУКУ

2.1. Порівняння мов програмування *JavaScript* та *Python*

Python – це інтерпретований мова програмування високого рівня з динамічною семантикою та об'єктно-орієнтованим програмуванням, розроблений так, щоб бути легким для читання та реалізації. Це мова сценаріїв, як Perl та Ruby, і використовується для створення веб-додатків теж. Це дозволяє програмістам використовувати різні стилі програм для простих і складних програм. Python підтримує різні парадигми програмування як об'єктно-орієнтоване програмування, функціональне програмування, імперативне програмування і процедурне програмування [6].

Java-Script – це об'єктно-орієнтована мова програмування, яка дозволяє створювати динамічні веб-сторінки та стандартизуватися в специфікації мови ECMAScript. Java-Script підтримує різні парадигми програмування як об'єктно-орієнтоване, функціональне і імперативне програмування, а не процедурне програмування. Він широко використовується в браузерах для забезпечення динамічної функціональності, яку ми не можемо досягти за допомогою звичайного HTML і CSS. Він підтримує стандартні програми з текстом, регулярними виразами та датою.

Python

- *REPL(Read-Eval-Print-Loop).* Ми отримаємо це з встановленням python і викликаємо різні версії python залежно від нашої установки.
- *Mutability.* Python має змінні і незмінні типи даних, такі як set (mutable) і список (Immutable).
- *Strings.* У Python вихідний код за замовчуванням ASCII, якщо ми не вказуємо будь-який формат кодування.

- *Numbers.* У Python ми маємо різні числові типи, такі як int, float, десяткове число з фіксованою точкою і таке інше.
- *Hash tables.* Python має вбудовані хеш-таблиці, які називаються словниками, наборами і т.д., які можна використовувати в хеші з ключами і значеннями.
- *Code Blocks.* Python використовує відступ.
- *Function arguments.* Тоді як Python покаже виняток, якщо функція викликається з неправильними параметрами і приймає деякий додатковий синтаксис передачі параметрів.
- *Data types.* Тоді як Python має два подібних списки типів даних і кортеж. Список Python і масив Java-Script дуже схожі.
- *Properties and Attributes.* Python дозволяє визначити атрибут, використовуючи протокол дескриптора, де ми можемо використовувати функції getter, setter.
- *Modules.* Python називає себе мовою, що включає батарею, оскільки вона постачається з широким спектром модулів.

JavaScript

- *REPL(Read-Eval-Print-Loop).* Не має вбудованого REPL, оскільки більшу частину часу ми виконуємо в браузері. Але ми можемо використовувати REPL шляхом встановлення node.js.
- *Mutability.* В той час як JavaScript не має поняття змінного і незмінного.
- *Strings.* В той час, як JavaScript повинен бути закодований як UTF-16 і немає вбудованої підтримки для маніпулювання сирими байтами.
- *Numbers.* В той час, як JavaScript містить лише числа з плаваючою точкою.

- *Hash tables.* В той час як Javascript не має вбудованої підтримки хеш-таблиці.
- *Code Blocks.* Тоді як Javascript використовує фігурні дужки.
- *Function arguments.* JavaScript не піклується про те, чи функції, викликані точними параметрами, не є типовими, оскільки будь-який відсутній параметр отримує значення "undefined", а будь-які додаткові аргументи закінчуються як спеціальні аргументи;
- *Data types.* JavaScript має вбудований тип масиву.
- *Properties and Attributes.* В той час як об'єкти JavaScript мають властивості, які можуть складатися з базових атрибутів, і це дозволяє визначити властивість.
- *Modules.* В той час, як JavaScript постачається з дуже невеликою кількістю модулів, таких як дата, математика, регулярні вирази, JSON, а також функціональність, доступна через хост-середовище, наприклад веб-браузер або інше середовище.

Різниця між мовами Python і JavaScript дуже мінімальна, тому що ми можемо робити майже все з мовами Python і JavaScript, оскільки вони поділяють спільні речі, такі як лексично масштабовані, об'єктно-орієнтовані, інтерпретовані, функціональні та імперативні. Зрештою, було обрано мову програмування Python. Адже у нас є багато існуючих бібліотек та інших розширень, з якими ми можемо досягти того, що нам необхідно [8].

2.2. Порівняння бібліотек для NLP

2.2.1. NLTK

Це найвідоміша бібліотека NLP Python, що призвела до неймовірних проривів у цій галузі. NLTK відповідає за завоювання багатьох проблем аналізу тексту.

NLTK також популярний для освіти та досліджень. На своєму власному сайті, NLTK стверджує, що це "дивовижна бібліотека, щоб грати з природною мовою".

NLTK має понад 50 корпусів і лексиконів, 9 створінь і десятки алгоритмів. Це тематичний парк наукового співробітника. Проте, це також є одним з основних недоліків NLTK. Він важкий і слизький, і він має круту криву навчання. Друга велика слабкість полягає в тому, що вона повільна і не готова до виробництва.

2.2.2. TextBlob

TextBlob сидить на могутніх плечах NLTK. TextBlob робить обробку тексту простим, надаючи інтуїтивно зрозумілий інтерфейс для NLTK. Це вітається доповненням до вже міцної лінійки бібліотек Python NLP, тому що вона має невисоку криву навчання, водночас дивовижною функціональністю. TextBlob це проста бібліотека, яка робить аналіз тексту радістю.

2.2.3. Stanford CoreNLP

Стенфордський CoreNLP являє собою набір готових до виробництва природних засобів аналізу. Вона включає в себе теги частин мови (POS), розпізнавання об'єктів, навчання шаблонів, синтаксичний аналіз і багато іншого. Фактично написаний на Java, а не на Python. Багато організацій використовують CoreNLP для реалізації продукції. Це швидкий, точний і здатний підтримувати кілька основних мов.

2.2.4. spaCy

Вона представлена як "промислова сила" бібліотеки NLP для Python, яка спрямована на продуктивність.

SpaCy є мінімальним і упізнаним, і не затоплює вас з параметрами, як NLTK. ЇЇ філософія полягає в тому, щоб представити лише один алгоритм (найкращий) для кожної мети. Вам не потрібно робити вибір, і ви можете зосередитися на продуктивності.

SpaCy називають «найсучаснішим», і важко не погодитися. Її головна слабкість полягає в тому, що вона наразі підтримує тільки англійську мову.

SpaCy є новою, тому її спільнота підтримки не така велика, як деякі інші бібліотеки. Тим не менш, його підхід до NLP є настільки переконливим, що це може привести до зриву NLTK.

2.2.5. Gensim

Gensim не для всіх, але те, що він робить, він робить добре. Gensim – це оптимізована бібліотека для моделювання тем і аналізу подібності документів. Серед перелічених тут бібліотек Python NLP це найбільш спеціалізовані.

Його алгоритми моделювання тематики, такі як реалізація Latent Dirichlet Allocation (LDA), є кращими у своєму класі. Крім того, він надійний, ефективний і масштабований.

Крім того, аналіз семантики суб-поля (або моделювання тематики) є однією з найбільш цікавих областей сучасної обробки природної мови.

Отже, проаналізувавши бібліотеки Python для NLP було прийнято рішення використовувати бібліотеку SpaCy.

2.3. Аналіз способів виконання семантичного пошуку даних

Існують різні підходи до організації семантичного пошуку за текстами. В останні роки найбільш популярним стало семантичне анотування тексту. Існують різні способи вирішення завдання семантичного анотування.

У кожному з них документу або частини документа приписується деякий набір семантично близьких документу міток. Надалі можна шукати документи

за цими мітках. Крім того, можна шукати документи звичайним повнотекстових пошуком, а потім враховувати ці мітки при роботі з документом, отримуючи більше інформації за допомогою них.

Зазвичай в якості міток використовуються персони, місця, організації або інші суб'єкти. Для опису міток часто використовуються RDF сховища, що містять набір понять і відносини між ними. Деякі методи використовують інформацію з Wikipedia, як з масштабного джерела знань.

Семантичне анотування не єдиний спосіб організації пошуку. Існують рішення, засновані на поліпшенні класичного повнотекстового пошуку з розширеним запитом синонімів. Крім того, підхід, використовує інформацію про синтаксис, морфології і пунктуації.

Було проведено безліч експериментів по використанню словників синонімів і гіпонімії для поліпшення якості повнотекстового пошуку.

Відомо, що при використанні синонімів і гіпонімії зростає повнота і часто істотно падає точність пошуку.

Особливість запропонованого підходу в тому, що індексується не весь текст, а тільки його значущі частини (в залежності від типу неоднорідних даних, веб-документи можуть бути частково розмічені, великі куски тексту – ні), в залежності від завдання, це можуть бути абзаци, речення, словосполучення або проставлена людиною мітка, наприклад, хештег. За рахунок зміни розміру значної частини можна контролювати точність і повноту.

Загальний алгоритм побудови системи семантичного Інтернет-пошуку (SWSS – Semantic Web Search System).

Загальний алгоритм роботи системи семантичного пошуку, який об'єднує кілька технологій і дозволяє створити мета-движок для фільтрації SWD-документів, що повертаються пошуковим механізмом Swoogle. Swoogle

– це пошукова система, заснована на індексування пошуковим роботом SWD-документів.

Swoogle надає методи для семантично пов'язаних документів до виконання запитів. Вона витягує з них метадані і підраховує залежності між документами.

Хоча Swoogle в даний час служить в якості SWD-індексується системи, яка використовується пошукова технологія заснована на пошуку лексичного відповідності термінів запиту і проіндексованих назв онтологічних класів і властивостей. Метою використання Swoogle є доказ того, що точність пошуку для простого запиту може бути поліпшена, якщо застосовується запропонований метод семантичного пошуку.

Крок 1. Переформулювання запиту. На цьому кроці для кожного терміна вільного запиту усувається неоднозначність, оцінюється передбачуване значення, яке визначається смисловим значенням WordNet.

Крок 2. Отримання і ранжування SWD-документів. На цьому кроці алгоритм переупорядковує SWD-документи, отримані з пошукової системи Swoogle. Повторне ранжування засноване на відкритій семантиці термінів запиту, переформулювати в онтологію запиту, і семантиці отриманих SWD-документів.

Спочатку призначений для користувача запит у вільній формі направляється Swoogle для отримання всіх доступних проіндексованих SWD-документів (онтологій). всі отримані SWD-документи відображаються на онтологію запиту, створену на кроці 1.

Переранжування грунтується на семантичній релевантності отриманих SWD-документів і онтології запиту. Зокрема, релевантність між двома онтологіями визначається як число їх концептів, для яких було знайдено відображення. Є інтуїтивно зрозумілим, що чим вище число відображених концептів, тим вище релевантність між онтологією запиту і отриманими SWD-

документами. Отримані SWD-документи з більш високою релевантність по відношенню до онтології запиту знаходяться вище в результуючому списку.

Проблеми семантичного пошуку.

Хоча семантична павутина сприяє пошуку інформації в мережі, існує кілька невирішених проблем, які слід взяти до уваги. Перша з них – це величезна кількість неструктурованих Інтернет-документів, які повинні бути семантично розмічені для використання семантичними пошуковими системами. Це не проста задача, вона, серед іншого, вимагає розвитку проблемно-орієнтованих онтологій.

Повністю автоматизований процес розмітки існуючих даних – ще одна невирішена задача. З іншого боку, ефективний пошук Інтернет-документів вимагає створення формальних запитів. Виходить, що звичайні користувачі Інтернету повинні вивчити формальну мову для створення такого роду запитів, а це не так просто.

Побудова відображення онтологій предметних областей на формальні запити також активно досліджується. Крім того, при розробці та реалізації семантичних пошукових систем виникає ще ряд проблем, які вказані нижче.

1. Використання зовнішніх ресурсів. SWSS повинна включати додаткові / зовнішні ресурси, якщо для них використовуються запити на природній мові. Таке знання може бути представлено у вигляді загальних словників або (і) у формі пов'язаної онтології.
2. Автоматизація та прозорість. SWSS забезпечує повністю прозорий процес пошуку для кінцевого користувача, передаючи ранжований список SWD-документів, які знаходить запит. Пошук SWD-документів здійснюється з мінімумом людського втручання.
3. Продуктивність. Пошук SWD-документів повинен виконуватися швидко. Час відгуку запиту в режимі реального часу в таких системах,

як семантична павутина, має велике значення. Таким чином, SWSS повинен бути реалізований як багатокроковий процес з коротким часом виконання кожного кроку для отримання бажаного результату.

4. Точність / повнота. Точність SWSS-системи – теж важливе питання. щоб пристрій запитував SWD-документів в реальних системах використовуються технології і реалізації повинні бути протестовані і оцінені в контексті точності і повноти одержуваного результату. Зокрема, автоматичне усунення неоднозначності термінів (автоматичне присвоєння смислових значень термінів) і автоматичний пошук SWD-документів (відображення онтології запиту на SWD-документи) забезпечують більш високу точність і повноту.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Структура програмного модуля

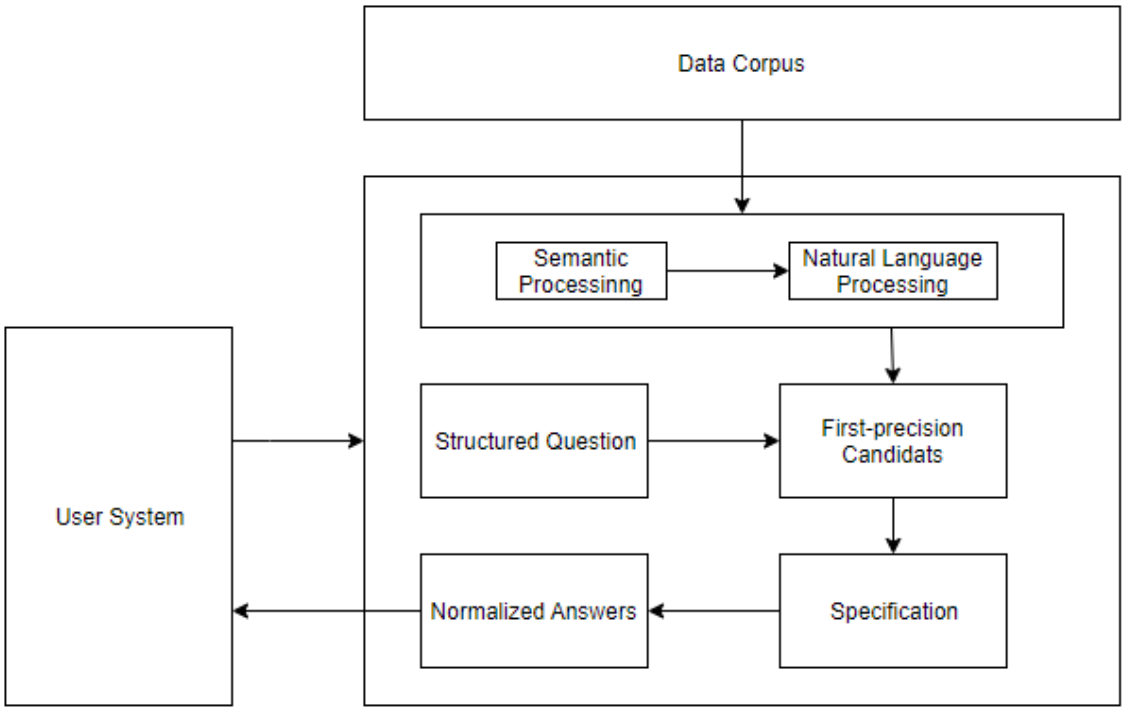


Рис. 3. Структура програмного модуля

Оглянемо основні структурні компоненти (модулі) програмного застосунку на прикладі спрощеної структурної схеми. Повна схема наведена у Додатках.

- **Structured Question.** Щоб підібрати сторінки, які містять релевантні теми, відповіді які потрібні користувачу, перш за все потрібно проаналізувати значення слів у контексті. Тому на вхід до головного модуля будуть надходити ключові слова. Тобто, користувач, відправив своє питання, це питання отримало обробку за допомогою *Natural Language Processing* і вже у вигляді ключових слів надходить до головного модуля. Модуль, який виконує аналіз запиту природною

мовою має на меті аналіз текстового представлення питання, яке задається користувачем, видалення з нього нерелевантних для пошуку відповіді слів, та, навпаки, виділення ключових слів, за якими можна було б провести пошук.

```
Structured Question:
{
  'python': 0.91;
  'def': 0.88;
  'function': 0.76;
  'self': 0.54;
  'this': 0.53;
  'comparison': 0.51;
}

Process finished with exit code 0
```

Рис. 4. Отримане структуроване питання

Отже, на вхід до нашого модуля семантичного пошуку приходить об’єкт в з ключовими словами та вагою цих слів, щоб задовольнити і покращити умови семантичного пошуку у корпусі неоднорідних даних.

- **Data Corpus.** Щоб задовольнити семантичний пошук по документу необхідно заздалегідь встановити відносини між символами і

об'єктами, які вони позначають. Можна виділити два основних види семантичного пошуку;

- **Повнотекстовий пошук.** Пошук по всьому вмісту документа з використанням попередньо побудованих індексів;
- **Пошук по метаданих.** Пошук за деякими атрибутами документа, які описують певні об'єкти підтримувані системою, наприклад, автор, адреса, назва організації і т. д.

Саме використання обох цих методів у поєднанні буде давати нам високий приріст у швидкості знаходження релевантної інформації. Пошук та індексація по метаданих буде широко використовуватись у даному модулі для семантичного пошуку в неоднорідному корпусі даних. Основний акцент концепції робиться на роботі з метаданими, які однозначно характеризують властивості і зміст веб-ресурсів, замість використовуваного в даний час текстового аналізу документів. Пошук буде здійснюватися за ключовими словами, що забезпечують доступ до мільярдів індексованих Інтернет-сторінок для мільярдів користувачів.

Отже, корпус неоднорідних даних представлений у вигляді *HTML* сторінок, які наповнені власними тегами та атрибутами для зрозумілого відображення для кінцевого користувача. Але щоб дістати корисну інформацію, модулю семантичного пошуку потрібно знайти спеціальні теги (найчастіше це <p>, <i>, <b>, і тд.) та дістати релевантну інформацію, стосовно нашого структурованого питання.

```
Structured Question:
{
  'python': 0.91;
  'def': 0.88;
  'function': 0.76;
  'self': 0.54;
  'this': 0.53;
  'comparison': 0.51;
}
Data Corpus:
{<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Python (programming language) - Wikipedia</title>
...
}

Process finished with exit code 0
```

Рис. 5. Отриманий корпус неоднорідних даних, відносно отриманого на вхід структурованого питання

3.2. Алгоритм семантичного пошуку

Рис. 6. Алгоритм семантичного пошуку

Завантаження даних

Відбувається шляхом завантаження проіндексованого корпусу неоднорідних даних. Корпус неоднорідних даних завантажується відносно структурованого питання, яке прийшло на вхід до головного модуля.

```
Data Corpus:
{<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Python (programming language) - Wikipedia</title>
...
}
```

Рис. 7. Завантажений корпус неоднорідних даних

Отримання ієрархії і ключових слів. Формування індексів

Для оптимального та більш швидкого пошуку по корпусу неоднорідних даних було проведено індексацію кожного веб-документа. Щоб прискорити процес індексації, даний модуль індексував документи виключно по

метаданим документа. З одного боку це дає великий приріст у швидкості індексації, з іншого боку, індексація по метаданим не завжди охоплює весь контент веб-документу.

```
url(https://en.wikipedia.org/wiki/Python_(programming_language))
indexing document:
{
  'python': 0.20;
  'api': 0.20;
  'methods': 0.20;
  'typing': 0.20;
  'history': 0.20;
}

Process finished with exit code 0
```

Рис. 8. Індексування документа

Пошук

Для оптимальної роботи семантичного пошуку в корпусі неоднорідних даних було для кожного домену свій алгоритм семантичного пошуку.

Тобто, візьмемо за приклад *en.wikipedia.org*, щоб знайти більш релевантну інформацію. Потрібно проходити по спеціальним блокам, які вже визначені наперед.

```
▼<p> == $0
  <b>Python</b>
  " is an "
  <a href="/wiki/Interpreted_language" title="Interpreted language">interpreted</a>
  ", "
  <a href="/wiki/High-level_programming_language" title="High-level programming language">high-level</a>
  ", "
  <a href="/wiki/General-purpose_programming_language" title="General-purpose programming language">general-purpose</a>
  <a href="/wiki/Programming_language" title="Programming language">programming language</a>
  ". Created by "
  <a href="/wiki/Guido_van_Rossum" title="Guido van Rossum">Guido van Rossum</a>
  " and first released in 1991, Python's design philosophy emphasizes "
  <a href="/wiki/Code_readability" class="mw-redirect" title="Code readability">code readability</a>
  " with its notable use of "
  <a href="/wiki/Off-side_rule" title="Off-side rule">significant whitespace</a>
  ". Its language constructs and "
  <a href="/wiki/Object-oriented_programming" title="Object-oriented programming">object-oriented</a>
  " approach aims to help programmers write clear, logical code for small and large-scale projects."
▶<sup id="cite_ref-AutoNT-7_27-0" class="reference">…</sup>
</p>
```

Рис. 9. Знаходження відповідних блоків з релевантною інформацією

Далі модуль збирає текст і формує готові параграфи, але вже без вкладених тегів на атрибутів (<a href="…">, <sup> , <b> і тд.). На виході ми отримаємо готові параграфи тексту.

```
url(https://en.wikipedia.org/wiki/Python_(programming_language))
Python is an interpreted, high-level, general-purpose programming language...
Python is dynamically typed and garbage-collected. It supports multiple programming...

Process finished with exit code 0
```

Рис. 10. Результат роботи семантичного пошуку

3.3. Алгоритм обробки природної мови

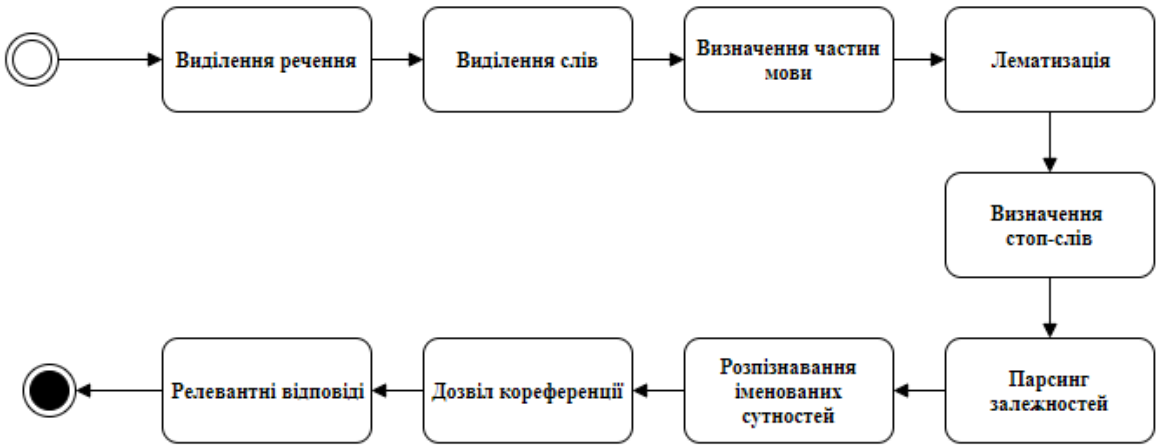


Рис. 11. Алгоритм обробки природної мови

Велика частина інформації не структурована – це просто тексти англійською або будь-якою іншою мовою. Цією проблемою займається особливий напрямок штучного інтелекту: обробка природної мови, або NLP (Natural Language Processing).

- *Виділення речення.* Оскільки на виході семантичного пошуку в корпусі неоднорідних даних ми маємо великі фрагменти тексту, доцільно буде поділити параграфи тексту на речення. Адже навчити розуміти модуль

для обробки природної мови тільки одне речення набагато простіше та швидше.

Можна було б просто розділяти текст за певними знаками пунктуації. Але сучасні NLP мають в запасі складніші методи, які підходять навіть для роботи з неформатованими фрагментами тексту.

- *Токенізація, або виділення слів.* Розділові знаки теж є токенами, оскільки можуть мати важливе значення.
- *Визначення частин мови.* Далі аналізуємо кожне слово разом з його найближчим оточенням за допомогою заздалегідь підготовленої класифікаційної моделі.

  Ця модель була навчена на мільйонах англійських речень з уже визначеними частинами мови для кожного слова і тепер здатна їх розпізнавати.

  Але цей аналіз заснований на статистиці – насправді модель не розуміє сенсу слів, вкладеного в них людиною. Вона просто знає, як вгадати частина мови, грунтуючись на схожу структуру речень, які були вивчені раніше.
- *Лематизація.* В англійській і більшості інших мовах слова можуть мати різні форми. У NLP цей процес називається лематизація – знаходженням основної форми (леми) кожного слова в реченні. Те ж саме відноситься до дієслів. Ми можемо привести їх до невизначеної форми. Лематизація зазвичай виконується простим пошуком форм в таблиці.
- *Визначення стоп-слів.* Тепер визначаємо важливість кожного слова в реченні. В англійській мові дуже багато допоміжних слів, наприклад, «and», «the», «a». При статистичному аналізі тексту ці маркери створюють багато шуму, так як з'являються частіше, ніж інші. Деякі NLP відзначають їх як стоп-слова і відсівають перед підрахунком

кількості. Для виявлення стоп-слів зазвичай використовуються готові таблиці. Однак немає єдиного стандартного списку, відповідного в будь-якій ситуації [17].

- *Парсинг залежностей.* Тепер встановлюємо взаємозв'язок між словами в реченні. Це називається парсинг залежностей. Кінцева мета цього кроку – побудова дерева, в якому кожен токен має єдиного батька. Коренем може бути головне дієслово. Також потрібно не тільки визначити батька, але і встановити тип зв'язку між двома словами. Також важливо пам'ятати, що багато англійських речень неоднозначні і складні для аналізу. У таких випадках модель робить припущення про найбільш ймовірне значення, хоча це не завжди виходить.
- *Розпізнавання іменованих сутностей (Named Entity Recognition, NER).* Мета розпізнавання іменованих сутностей – виявити такі іменники і зв'язати їх з реальними концепціями.

  NER-системи не просто переглядає словники. Вона аналізує контекст токена в реченні і використовує статистичні моделі, щоб вгадати який об'єкт він представляє.

  Більшість NER-моделей розпізнають наступні типи об'єктів:

  - імена людей;
  - назви компаній;
  - географічні позначення (і фізичні, і політичні);
  - продукти;
  - дати і час;
  - грошові суми;
  - події.
- *Дозвіл кореференції*. В англійському дуже багато займенників – слів he, she, it. Це скорочення, якими ми замінюємо на листі справжні імена і назви. Людина може простежити взаємозв'язок цих слів у тексті,

грунтуючись на контексті. Але дана NLP-модель не знає про те, що означають займенники, адже вона розглядає лише одне речення за раз. Дозволом кореференції називається відстеження займенників в тексті з метою вибрати всі слова, що відносяться до однієї сутності.

Скомбінувавши цю методику з деревом парсинга та інформацією про іменовані сутності, ми отримуємо можливість витягти з документа величезну кількість корисних даних.

4. АНАЛІЗ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

4.1. Особливості реалізації модуля

Особливість реалізація семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних полягає у тому, що цей модуль може бути підключений до будь-якого мессенджера, або соціальної мережі, де є можливість використовувати бота.

Даний модуль значно прискорює пошук релевантної інформації в певній предметній області та надає декілька відповідей на структуроване питання, чим підтверджує свою корисність та швидкодію.

Семантичний пошук надає можливість шукати інформацію на просторах веб-документів, що значно підвищує шанс знаходження більш актуальної та корисної для користувача інформації.

Семантичний пошук працює з неоднорідним корпусом даних, тому для коректності та більш стислого об’єму кінцевої відповіді використовується обробка природної мови. Вона в свою чергу розбиває текст на речення та дістає головну смислову частину речення. Щоб користувачу було зручніше читати маленький та більш інформативний текстовий фрагмент ніж великий параграф тексту.

Однією з особливостей модуля семантичного пошука є індексація документа за його метаданими. Це значно прискорює його індексацію.

Для отримання корпусу неоднорідних даних у форматі HTML сторінок було використано бібліотеку *Beautiful Soup.*

Beautiful Soup – це бібліотека Python для виведення даних з файлів HTML і XML. Вона дуже добре підходить, щоб забезпечити ідіоматичні способи навігації, пошуку та модифікації дерева веб-сторінки.

Для роботи з Wikipedia API була використана бібліотека Python – *wikipedia*.

4.2. Тестування системи

- *Тестування 1*:
  - *Вхідні параметри*: структуроване питання;
  - *Крок*: структуроване питання потрапляє на вхід до модуля;
  - *Очікуваний результат*: структуроване питання відправиться на обробку до семантичного алгоритму, де проаналізувавши корпус неоднорідних даних буде виявлено найбільш відповідні документи.

    Далі за допомогою обробки штучної мови, буде знайдено найбільш доцільніші варіанти відповіді.

    З цих варіантів відповідей буде сформована загальна відповідь, яка піде на вихід користувачу даного модуля.

```
Structured Question:
{
  'python': 0.91;
  'def': 0.88;
  'function': 0.76;
  'self': 0.54;
  'this': 0.53;
  'comparison': 0.51;
}

Process finished with exit code 0
```

Рис. 12. Приклад роботи (результату) тестування 1

- *Тестування 2*:
  - *Вхідні параметри*: пуста рядок;
  - *Крок*: пуста рядок потрапляє на вхід до модуля;
  - *Очікуваний результат*: спрацьовує перевірка для визначення та розпізнавання структурованого питання.

    Дана перевірка повідомляє про помилку у запиті.

    Робота модуля зупиняється і користувачу надходить інформація про некоректність даних, з якими даний модуль не може працювати.

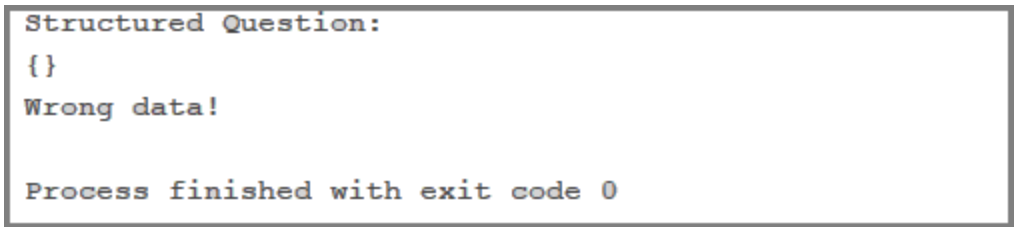


Рис. 13. Приклад роботи тестування 2

- *Тестування 3*:
  - *Вхідні параметри*: Структуроване питання.
  - *Крок*: Семантичний алгоритм у ході свого виконання не знайшов доцільних документів у корпусі неоднорідних даних.
  - *Очікуваний результат*: Спрацьовує перевірка. Робота модуля припиняється.

    На вихід користувачу надходить повідомлення, що відповіді на поставлене запитання не було знайдено.

```
Structured Question:
{
   'hello': 0.50;
   'world': 0.50;
}
The answer to the question was not found.

Process finished with exit code 0
```

Рис. 14. Приклад роботи тестування 3

4.3. Рекомендації до користування додатком (модулем)

Збільшення, оновлення, доповнення документів у корпусі неоднорідних даних.

У майбутньому NLP-модель буде вдосконалюватися і більш розумно обробляти тексти.

Вдосконалення семантичного пошуку:

- прискорити завантаження даних;
- збільшення продуктивності пошуку документів, шляхом швидкого отримання ієрархії документа;
- отримати швидкий підхід для знаходження ключових слів;
- збільшити швидкість у формуванні індексів.

Постійно оновлювати та збагачувати новими методами та технологіями обробку природної мови, щоб знайдена відповідь була максимально зрозуміла для користувача та несла за собою загальну відповідь на поставлене запитання.

ВИСНОВКИ

Метою даного дипломного проекту є розроблення програмного модуля для семантичного пошуку відповідей на структуровані питання в корпусі неоднорідних даних.

Проведений у рамках даного проекту аналіз показав, що така система має для виконання своєї прямої функції – семантичного пошуку – також обробку природної мови, та мати достатньо гнучку архітектуру, щоб мати змогу легко розширюватися при доданні нових джерел для поповнення корпусу неоднорідних даних.

Розроблений програмний модуль:

- аналізує структуроване питання;
- проводить семантичний пошук по корпусу неоднорідних даних:
  - завантажує дані;
  - отримує ієрархії і ключових слів;
  - формує індекси;
  - виконує пошук, безпосередньо семантичний пошук;
- проводить алгоритм оброблення природної мови.

Розробку виконано у повному обсязі. Усі вимоги, наведенні у технічному завданні, виконані. Програмний модуль, згідно з затвердженою методикою виконання, успішно протестована.

Використання даного модуля дозволить зменшити час, який користувач витрачає на знаходження інформації за допомогою пошуковиків у певній предметній області та значно полегшить роботу/взаємодію для студента та викладача.

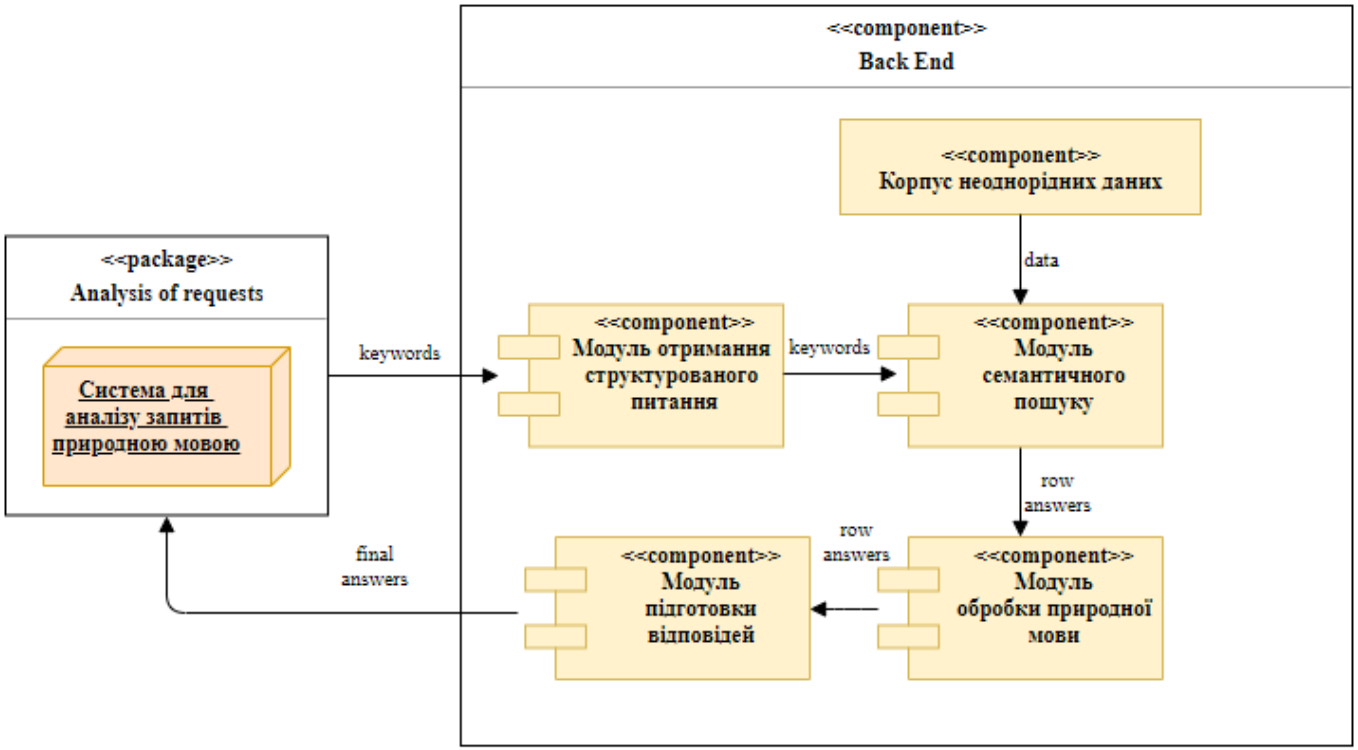
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Berners-Lee T. The Semantic Web [Електронний ресурс]. — Режим доступу: https://www.researchgate.net/publication/307845029_Tim_Berners-Lee's_Semantic_Web
2. Slack – Where Work Happens [Електронний ресурс]. — Режим доступу: https://slack.com/intl/en-ua/
3. Beautiful Soup is licensed under the same terms as Python itself [Електронний ресурс]. — Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
4. Jill Watson Doesn’t Care if You’re Pregnant: Grounding AI Ethics in Empirical Studies [Електронний ресурс]. — Режим доступу: http://www.aies-conference.com/wp-content/papers/main/AIES_2018_paper_104.pdf
5. A teaching assistant named Jill Watson [Електронний ресурс]. — Режим доступу: https://youtu.be/WbCguICyfTA?t=645
6. A teaching assistant named Jill Watson [Електронний ресурс]. — Режим доступу: https://youtu.be/WbCguICyfTA?t=330
7. Python – Wikipedia[Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Python_(programming_language)
8. Kim, Y. Convolutional Neural Networks for Sentence Classification [Електронний ресурс] / Yoon Kim. — New York University, 2014. — 6 p. Режим доступу: https://www.aclweb.org/anthology/D14-1181
9. LeCun, Y. Text Understanding from Scratch, request and beauty [Електронний ресурс] / Yann LeCun, Xiang Zhang. — New York University, 2016. — 10 p. Режим доступу: https://arxiv.org/pdf/1502.01710.pdf

10. Lai, S. Recurrent Convolutional Neural Networks for Text Classification [Електронний ресурс] / Siwei Lai, Liu Kang, Liheng Xu, Jun Zhao.— Institute of Automation, Chinese Academy of Sciences, 2015. — 7 p.
11. Beaufays, F. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling [Електронний ресурс] / Francoise Beaufays, Hasim Sak, Andrew Senior. — Google, 2014. — 5 p. Режим доступу: https://wiki.inf.ed.ac.uk/twiki/pub/CSTR/ListenTerm1201415/sak2.pdf
12. He L. Part-of-Speech Tagging [Електронний ресурс] / Lei He, Frank K. Soong, Peilu Wang, Hai Zhao. — Jiao Tong University; Microsoft Research Asian; Educational Testing Service Research, 2015. — 6 p. Режим доступу: https://arxiv.org/pdf/1510.06168.pdf
13. Quoc, V. L. Sequence to Sequence Learning with Neural Networks [Електронний ресурс] / V. Le Quoc, Ilya Sutskever, Oriol Vinyals. — Google, 2014. — 9 p. Режим доступу: https://arxiv.org/pdf/1409.3215.pdf
14. Huang, E. Paraphrase Detection Using Recursive Autoencoder [Електронний ресурс] / Eric Huang. — Stanford University, 2011. — 8 с. Режим доступу: https://nlp.stanford.edu/courses/cs224n/2011/reports/ehhuang.pdf
15. Chen, K. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] / Kai Chen, Greg Corrado, Jeffrey Dean, Tomas Mikolov. — Google, 2013. — 12 p. Режим доступу: https://arxiv.org/pdf/1301.3781.pdf
16. Stack Overflow – Where Developers Learn, Share, & Build Careers [Електронний ресурс]. — Режим доступу: https://stackoverflow.com/

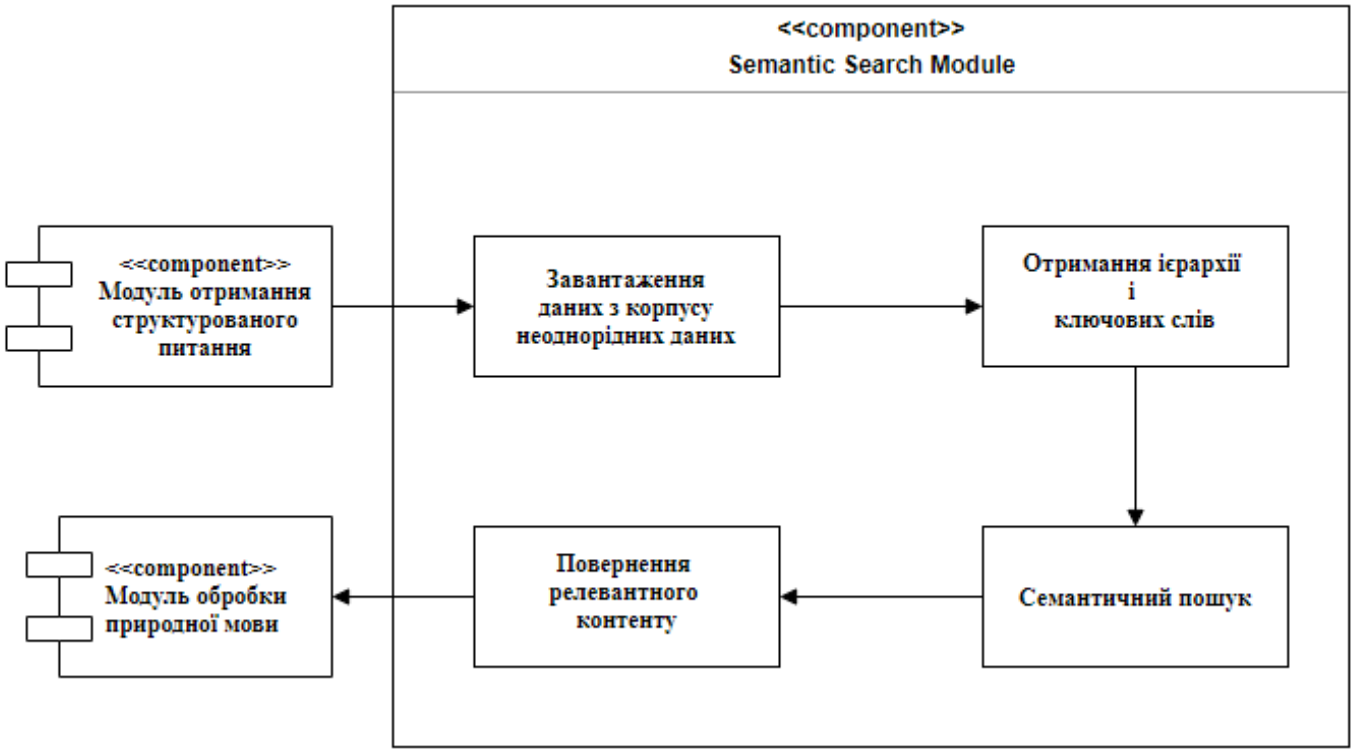
Додаток 1

Копії графічних матеріалів



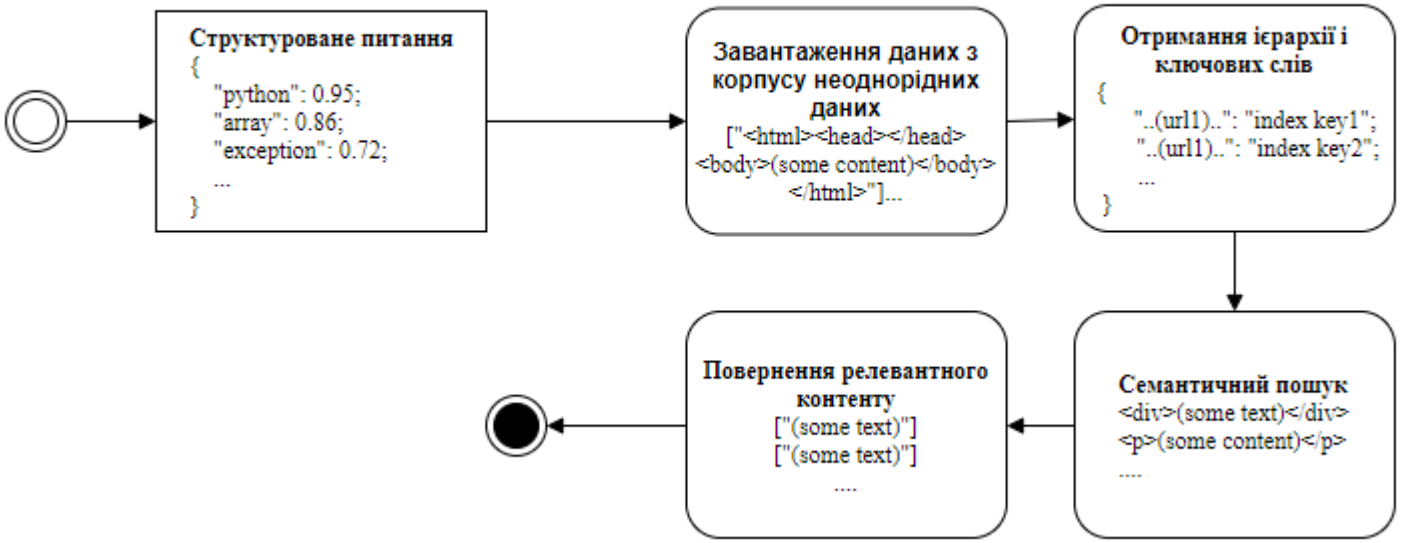
ДП.045440-06-99

Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних. Структурна схема програмної системи. UML діаграма компонентів



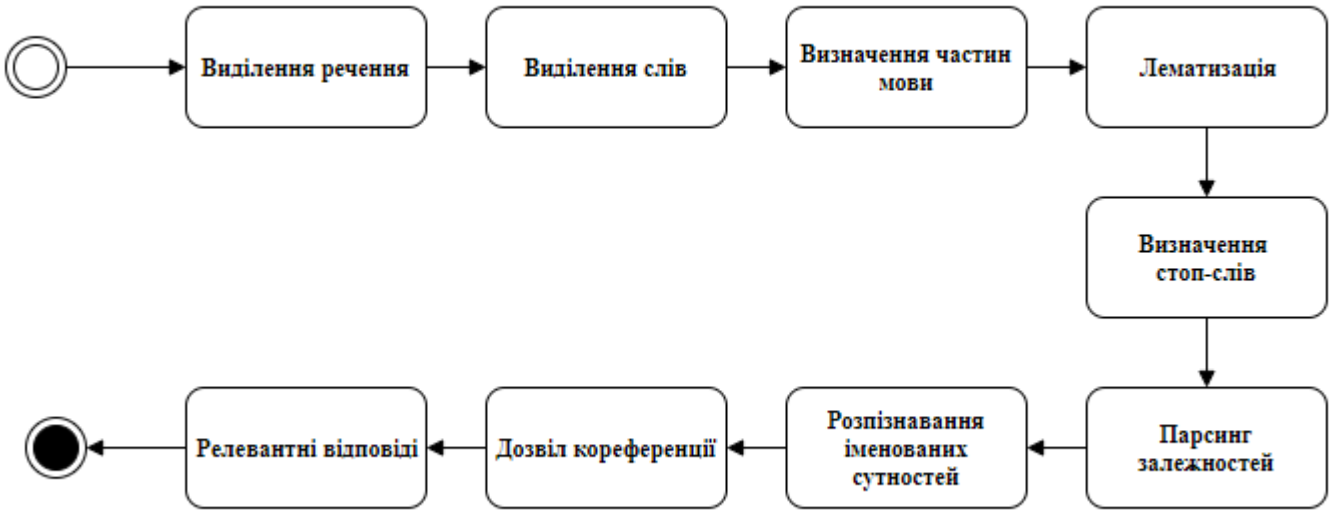
ДП.045440-07-99

Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних. Структурна схема семантичного пошуку. UML діаграма компонентів



Алгоритм семантичного пошуку у корпусі неоднорідних даних

Андрієнко Ф.О., група КП-51



Алгоритм обробки природної мови

Андрієнко Ф.О., група КП-51

Додаток 2

Лістинг програми

Модуль роботи з Wikipedia Api

```
import wikipediaapi
from wiki_itemXML import ItemXML


class ItemWikiApi:
    def __init__(self, categoryName, summary, url, title, text, pageId):
        self.categoryName = categoryName
        self.summary = summary
        self.url = url
        self.title = title
        self.text = text
        self.pageId = pageId



class WikiApi:
    def __init__(self, category):
        self.category = category
        self.wiki_wiki = wikipediaapi.Wikipedia('en')

    def print_sections(self, sections, summary, url, categoryName, pageId,
level=0):
        for s in sections:
            self.print_sections(s.sections, summary, url, categoryName,
pageId, level + 1)
            item = ItemWikiApi(categoryName, summary, url, s.title,
s.text, pageId)
            ItemXML().updateXML(item, self.filename)


    def print_categorymembers(self, categorymembers, level=0,
max_level=0):
        for c in categorymembers.values():
```

```
                self.filename = "xmls2/" + str(c.pageid) + '.xml'
                if self.filename.find("Category:") == -1:
                    print("Title: %s" % (c.title))
                    page_py = self.wiki_wiki.page(c.title)
                    print("URL: %s" % (page_py.fullurl))
                    ItemXML().createXML(str(self.filename))
                    self.print_sections(page_py.sections,   page_py.summary,
page_py.fullurl, c.title, str(c.pageid))

                if  c.ns  ==  wikipediaapi.Namespace.CATEGORY  and  level  <=
max_level:
                    self.print_categorymembers(c.categorymembers, level + 1)

        def main(self):
            cat = self.wiki_wiki.page(self.category)
            self.print_categorymembers(cat.categorymembers)
    class ItemXML():
        def createXML(self, filename):
            root = ET.Element("data")
            tree = ET.ElementTree(root)
            tree.write(filename, encoding='utf-8', xml_declaration=True)

        def updateXML(self, item, filename):
            tree = ET.parse(filename)
            root = tree.getroot()
            if root.find(".//page[@url='" +item.url+ "']"):
                fragment = root.find(".//page[@url='" +item.url+ "']")
                ET.SubElement(fragment,     "fragment",     title=item.title,
type="text").text = item.text
            else:
                fragment   =   ET.SubElement(root,   "page",   url=item.url,
name=item.categoryName, id=item.pageId)
```

```
                ET.SubElement(fragment,   "summary",   type="text").text   =
item.summary
                ET.SubElement(fragment,    "fragment",    title=item.title,
type="text").text = item.text
            tree = ET.ElementTree(root)
            tree.write(filename, encoding='utf-8', xml_declaration=True)
   def set_lang(prefix):
     '''
     Change the language of the API being requested.
     Set `prefix` to one of the two letter prefixes found on the `list of all
Wikipedias <http://meta.wikimedia.org/wiki/List_of_Wikipedias>`_.
     After setting the language, the cache for ``search``, ``suggest``, and
``summary`` will be cleared.
     .. note:: Make sure you search for page titles in the language that you
have set.
     '''
     global API_URL
     API_URL = 'http://' + prefix.lower() + '.wikipedia.org/w/api.php'

     for cached_func in (search, suggest, summary):
       cached_func.clear_cache()



   def set_user_agent(user_agent_string):
     '''
     Set the User-Agent string to be used for all requests.
     Arguments:
     * user_agent_string - (string) a string specifying the User-Agent header
     '''
     global USER_AGENT
     USER_AGENT = user_agent_string
```

```
   def set_rate_limiting(rate_limit, min_wait=timedelta(milliseconds=50)):
     '''
     Enable or disable rate limiting on requests to the Mediawiki servers.
     If rate limiting is not enabled, under some circumstances (depending on
     load on Wikipedia, the number of requests you and other `wikipedia` users
     are making, and other factors), Wikipedia may return an HTTP timeout error.
     Enabling rate limiting generally prevents that issue, but please note that
     HTTPTimeoutError still might be raised.
     Arguments:
     * rate_limit - (Boolean) whether to enable rate limiting or not
     Keyword arguments:
     * min_wait - if rate limiting is enabled, `min_wait` is a timedelta describing the minimum time to wait before requests.
            Defaults to timedelta(milliseconds=50)
     '''
     global RATE_LIMIT
     global RATE_LIMIT_MIN_WAIT
     global RATE_LIMIT_LAST_CALL

     RATE_LIMIT = rate_limit
     if not rate_limit:
       RATE_LIMIT_MIN_WAIT = None
     else:
       RATE_LIMIT_MIN_WAIT = min_wait

     RATE_LIMIT_LAST_CALL = None
```

```
  @cache
  def search(query, results=10, suggestion=False):
    '''
    Do a Wikipedia search for `query`.
    Keyword arguments:
    * results - the maxmimum number of results returned
    * suggestion - if True, return results and suggestion (if any) in a
tuple
    '''

    search_params = {
      'list': 'search',
      'srprop': '',
      'srlimit': results,
      'limit': results,
      'srsearch': query
    }
    if suggestion:
      search_params['srinfo'] = 'suggestion'

    raw_results = _wiki_request(search_params)

    if 'error' in raw_results:
      if raw_results['error']['info'] in ('HTTP request timed out.', 'Pool
queue is full'):
        raise HTTPTimeoutError(query)
      else:
        raise WikipediaException(raw_results['error']['info'])

    search_results = (d['title'] for d in raw_results['query']['search'])

    if suggestion:
```

```
      if raw_results['query'].get('searchinfo'):
        return list(search_results), raw_results['query']['searchinfo']['suggestion']
      else:
        return list(search_results), None

    return list(search_results)


  @cache
  def geosearch(latitude, longitude, title=None, results=10, radius=1000):
    '''
    Do a wikipedia geo search for `latitude` and `longitude`
    using HTTP API described in http://www.mediawiki.org/wiki/Extension:GeoData
    Arguments:
    * latitude (float or decimal.Decimal)
    * longitude (float or decimal.Decimal)
    Keyword arguments:
    * title - The title of an article to search for
    * results - the maximum number of results returned
    * radius - Search radius in meters. The value must be between 10 and 10000
    '''

    search_params = {
      'list': 'geosearch',
      'gsradius': radius,
      'gscoord': '{0}|{1}'.format(latitude, longitude),
      'gslimit': results
    }
    if title:
```

```
        search_params['titles'] = title

    raw_results = _wiki_request(search_params)

    if 'error' in raw_results:
      if raw_results['error']['info'] in ('HTTP request timed out.', 'Pool
queue is full'):
        raise HTTPTimeoutError('{0}|{1}'.format(latitude, longitude))
      else:
        raise WikipediaException(raw_results['error']['info'])

    search_pages = raw_results['query'].get('pages', None)
    if search_pages:
      search_results = (v['title'] for k, v in search_pages.items() if k !=
'-1')
    else:
      search_results       =       (d['title']       for       d       in
raw_results['query']['geosearch'])

    return list(search_results)



  @cache
  def suggest(query):
    '''
    Get a Wikipedia search suggestion for `query`.
    Returns a string or None if no suggestion was found.
    '''

    search_params = {
      'list': 'search',
      'srinfo': 'suggestion',
```

```
        'srprop': '',
      }
      search_params['srsearch'] = query

      raw_result = _wiki_request(search_params)

      if raw_result['query'].get('searchinfo'):
        return raw_result['query']['searchinfo']['suggestion']

      return None



   def random(pages=1):
      '''
      Get a list of random Wikipedia article titles.
      .. note:: Random only gets articles from namespace 0, meaning no
Category, User talk, or other meta-Wikipedia pages.
      Keyword arguments:
      * pages - the number of random pages returned (max of 10)
      '''

#http://en.wikipedia.org/w/api.php?action=query&list=random&rnlimit=5000&fo
rmat=jsonfm
      query_params = {
        'list': 'random',
        'rnnamespace': 0,
        'rnlimit': pages,
      }

      request = _wiki_request(query_params)
      titles = [page['title'] for page in request['query']['random']]
```

```
    if len(titles) == 1:
      return titles[0]

    return titles


  @cache
  def summary(title, sentences=0, chars=0, auto_suggest=True, redirect=True):
    '''
    Plain text summary of the page.
    .. note:: This is a convenience wrapper - auto_suggest and redirect are enabled by default
    Keyword arguments:
    * sentences - if set, return the first `sentences` sentences (can be no greater than 10).
    * chars - if set, return only the first `chars` characters (actual text returned may be slightly longer).
    * auto_suggest - let Wikipedia find a valid page title for the query
    * redirect - allow redirection without raising RedirectError
    '''

    # use auto_suggest and redirect to get the correct article
    # also, use page's error checking to raise DisambiguationError if necessary
    page_info = page(title, auto_suggest=auto_suggest, redirect=redirect)
    title = page_info.title
    pageid = page_info.pageid

    query_params = {
      'prop': 'extracts',
      'explaintext': '',
```

```
        'titles': title
      }

      if sentences:
        query_params['exsentences'] = sentences
      elif chars:
        query_params['exchars'] = chars
      else:
        query_params['exintro'] = ''

      request = _wiki_request(query_params)
      summary = request['query']['pages'][pageid]['extract']

      return summary


    def page(title=None, pageid=None, auto_suggest=True, redirect=True,
preload=False):
      '''
      Get a WikipediaPage object for the page with title `title` or the pageid
      `pageid` (mutually exclusive).
      Keyword arguments:
      * title - the title of the page to load
      * pageid - the numeric pageid of the page to load
      * auto_suggest - let Wikipedia find a valid page title for the query
      * redirect - allow redirection without raising RedirectError
      * preload - load content, summary, images, references, and links during
initialization
      '''

      if title is not None:
        if auto_suggest:
```

```
          results, suggestion = search(title, results=1, suggestion=True)
          try:
            title = suggestion or results[0]
          except IndexError:
            # if there is no suggestion or search results, the page doesn't
exist
            raise PageError(title)
        return WikipediaPage(title, redirect=redirect, preload=preload)
      elif pageid is not None:
        return WikipediaPage(pageid=pageid, preload=preload)
      else:
        raise ValueError("Either a title or a pageid must be specified")




    class WikipediaPage(object):
      '''
      Contains data from a Wikipedia page.
      Uses property methods to filter data from the raw HTML.
      '''

      def    __init__(self,    title=None,    pageid=None,    redirect=True,
preload=False, original_title=''):
        if title is not None:
          self.title = title
          self.original_title = original_title or title
        elif pageid is not None:
          self.pageid = pageid
        else:
          raise ValueError("Either a title or a pageid must be specified")

        self.__load(redirect=redirect, preload=preload)
```

```
        if preload:
          for prop in ('content', 'summary', 'images', 'references', 'links',
'sections'):
            getattr(self, prop)

      def __repr__(self):
        return stdout_encode(u'<WikipediaPage \'{}\'>'.format(self.title))

      def __eq__(self, other):
        try:
          return (
            self.pageid == other.pageid
            and self.title == other.title
            and self.url == other.url
          )
        except:
          return False

      def __load(self, redirect=True, preload=False):
        '''
        Load basic information from Wikipedia.
        Confirm that page exists and is not a disambiguation/redirect.
        Does not need to be called manually, should be called automatically
during __init__.
        '''
        query_params = {
          'prop': 'info|pageprops',
          'inprop': 'url',
          'ppprop': 'disambiguation',
          'redirects': '',
        }
```

```
if not getattr(self, 'pageid', None):
  query_params['titles'] = self.title
else:
  query_params['pageids'] = self.pageid;
```

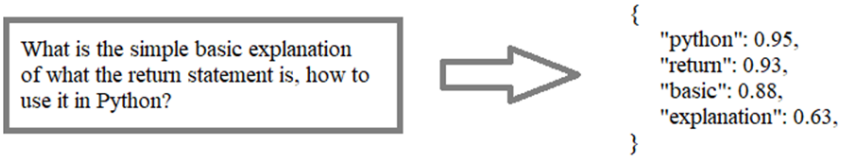
Додаток 3

Копії презентації

ПРОГРАМНИЙ МОДУЛЬ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ВІДПОВІДЕЙ НА СТРУКТУРОВАНІ ПИТАННЯ У КОРПУСІ НЕОДНОРІДНИХ ДАНИХ

Виконав:
студент Андрієнко Ф.О.
ФПМ, КП-51
Науковий керівник:
доцент, к.т.н., Заболотня Т.М.

Що таке Структуроване питання?



Постановка задачі

Мета:

Побудувати модуль, для пошуку у веб-мережі найбільш релевантної та корисної інформації у межах певної предметної дисципліни.

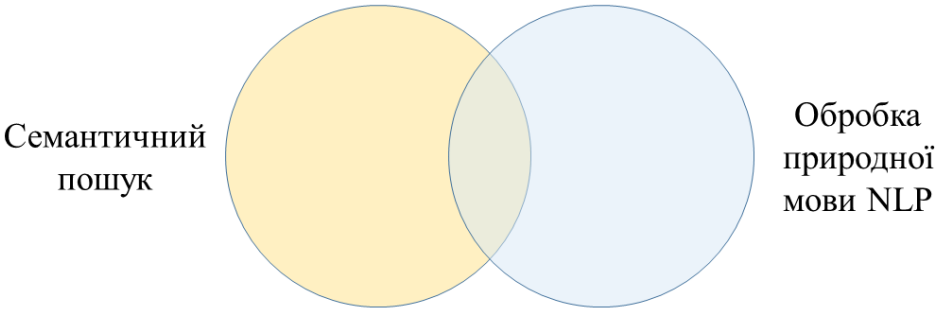
Відповідні задачі, які необхідно вирішити:

- реалізувати алгоритм семантичного пошуку;
- реалізувати алгоритм пошуку релевантних відповідей;
- виконати тестування програмного модуля.

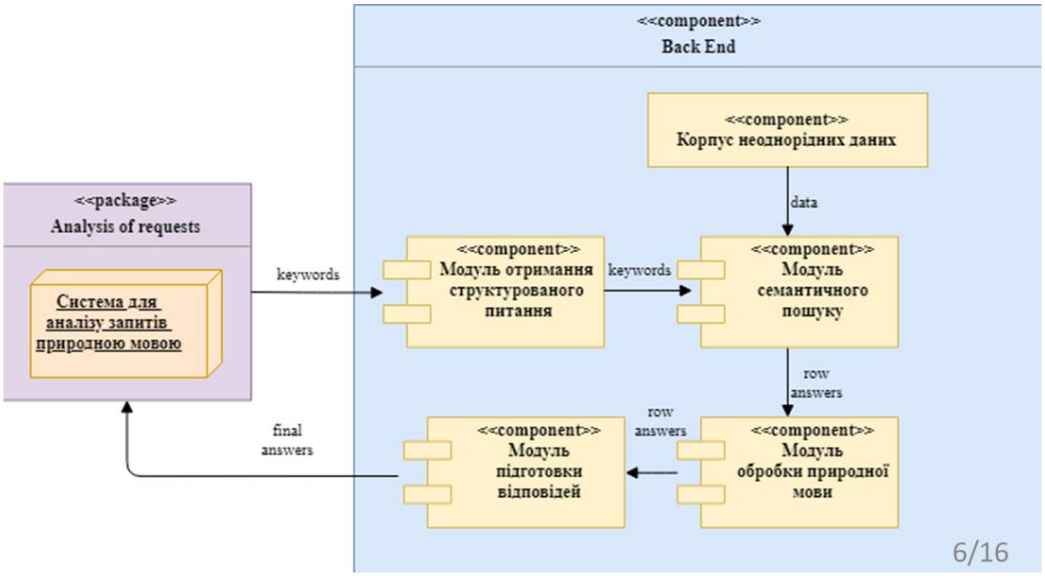
Актуальність

- Полегшення пошуку інформації на питання певної спеціалізації.

Особливості побудованого модуля



Архітектура системи



Семантичний пошук

NLP

Як інформація представлена в інтернет ресурсі

Корпус неоднорідних даних

```
▼<h1 id="firstHeading" class="firstHeading" lang="en">
    ::before
    "Python (programming language)"
  </h1>
▼<div id="bodyContent" class="mw-body-content">
    <div id="siteSub" class="noprint">From Wikipedia, the free encyclopedia</div>
    <div id="contentSub"></div>
    <div id="jump-to-nav"></div>
    <a class="mw-jump-link" href="#mw-head">Jump to navigation</a>
    <a class="mw-jump-link" href="#p-search">Jump to search</a>
  ▼<div id="mw-content-text" lang="en" dir="ltr" class="mw-content-ltr">
    ▼<div class="mw-parser-output">
      ▶<div role="note" class="hatnote navigation-not-searchable">…</div>
        <div class="shortdescription nomobile noexcerpt noprint searchaux" style=
        "display:none">General-purpose, high-level programming language</div>
        <p class="mw-empty-elt">
        </p>
```

Технології розробки

- Мова програмування: ***Python;***
- Бібліотека NLP: ***SpaCy;***
- Інші бібліотеки: ***Scrapy, BeautifulSoup.***

Приклад роботи модуля

```
Structured Question:
{
  'python': 0.93;
  'main': 0.91;
  'info': 0.86;
}
Answers:
Python is an interpreted, high-level, general-purpose programming language
Python is dynamically typed and garbage-collected
Python interpreters are available for many operating systems
RunTimer: 00:00:12.500232

Process finished with exit code 0
```

Результати тестування з заміром часу

Спроба	Час
1	00:00:12.500232
2	00:00:13.483112
3	00:00:11.791347
4	00:00:10.235487
5	00:00:10.897320

Платформи для прикріплення модуля

- Telegram
- Viber
- Messenger
- Slack

Висновки

Було реалізовано:

- алгоритм семантичного пошуку;
- алгоритм пошуку релевантних відповідей за допомогою (NLP);

Було проведено тестування модуля.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ПРОГРАМНИЙ МОДУЛЬ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ВІДПОВІДЕЙ НА СТРУКТУРОВАНІ ПИТАННЯ У КОРПУСІ НЕОДНОРІДНИХ ДАНИХ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Ф.О. Андрієнко

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмний модуль для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних являє собою сервер, написаний на мові програмування *Python*. Доступ до функціоналу забезпечується через програмну систему для аналізу засобів природної мовою.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Коректність вхідних даних (структурованого питання);
2. Взаємодія сервера з іншими опціональними модулями;
3. Забезпечення коректної обробки запитів;
4. Забезпечення коректного семантичного пошуку;
5. Відповідність системи вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується комбінацією методів структурного тестування та методів поведінкового тестування. Такий підхід також називається тестуванням за принципом “сірої скриньки”. Перевіряється як сама кодова база, так і відповідний програмний продукт під час інтеракції на відповідність функціональним вимогам. Тестування відбувається на повній інтегрованій системі, тобто на рівні системного тестування.

Використовуються наступні методи:

1. Функціональне тестування, в тому числі тестування критичного шляху (тестування системи за стандартних умов користування нею);

2. Димове тестування;
3. Тестування взаємодії, в тому числі тестування сумісності та інтеграційне тестування.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність створеної системи перевіряється шляхом:

1. Динамічного ручного тестування через введення некоректних вхідних даних (структурованого питання);
2. Динамічного ручного тестування на відповідність функціональним вимогам;
3. Статичного тестування кодової бази;
4. Тестування при конкурентному навантажені на систему;
5. Тестування стабільності роботи за різних умов.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ І.А.Дичка

"___" _____ 2019 р.

ПРОГРАМНИЙ МОДУЛЬ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ВІДПОВІДЕЙ НА СТРУКТУРОВАНІ ПИТАННЯ У КОРПУСІ НЕОДНОРІДНИХ ДАНИХ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В.Онай

Виконавець:

_____Ф.О. Андрієнко

2019

ЗМІСТ

1. Загальні відомості про систему

Дана система представляє собою програмне забезпечення, яке можна використовувати для семантичного пошуку відповідей на структуровані питання у корпусі неоднорідних даних.

Інтерактування з системою відбувається через інший модуль, який описує взаємодію з користувачем за використанням боту, розміщеного на платформі *Telegram.*

Загалом взаємодія з модулем відбувається в мінімальному режеми за допомогою консолі.

2. Процедура передачі структурованого питання до модуля

У даному пункті передбачається, що користувач добре розуміє функціонал системи та вміє користуватися консоллю. Якщо ж ні, то потрібно використовувати модуль, який описує взаємодію з користувачем за використанням боту.

Для того, щоб передати структуроване питання та запустити процес семантичного пошуку потрібно запустити модуль за допомогою Python інтерпретатора.

Далі потрібно викликати метод “enteredStructQuestion()”. Далі пишемо структуроване питання у правильному форматі.

Щоб запустити процес семантичного пошуку необхідно викликати метод “startSemanticSearch()”.

3. Отримання кінцевого результату

Отриманий результат відображається у консолі у вигляді об’єкту. Щоб завантажити і зберегти цей об’єкт до файлу необхідно викликати метод “saveToFile()”.